Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 109781, 9 pages
http://dx.doi.org/10.1155/2013/109781

Research Article

An Efficient Schema for Cloud Systems Based on SSD Cache Technology

Jinjiang Liu,[1] Yihua Lan,[1] Jingjing Liang,[1] Quanzhou Cheng,[1] Chih-Cheng Hung,[2,3] Chao Yin,[4] and Jiadong Sun[4]

[1] *School of Computer and Information Technology, Nanyang Normal University, Nanyang 473061, China*
[2] *Sino-US Intelligent Information Processing Joint Lab, Anyang Normal University, Anyang 455000, China*
[3] *Center for Biometrics Research, Southern Polytechnic State University, Atlanta, GA 30060-2896, USA*
[4] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

Correspondence should be addressed to Yihua Lan; lanhua_2000@sina.com

Received 11 July 2013; Accepted 7 September 2013

Academic Editor: William Guo

Traditional caching strategy is mainly based on the memory cache, taking read-write speed as its ultimate goal. However, with the emergence of SSD, the design ideas of traditional cache are no longer applicable. Considering the read-write characteristics and times limit of erasing, the characteristics of SSD are taken into account as far as possible at the same time of designing caching strategy. In this paper, the flexible and adaptive cache strategy based on SSD is proposed, called FAC, which gives full consideration to the characteristics of SSD itself, combines traditional caching strategy design ideas, and then maximizes the role SSD has played. The core mechanism is based on the dynamic adjustment capabilities of access patterns and the efficient selection algorithm of hot data. We have developed dynamical adjust section hot data algorithm, DASH in short, to adjust the read-write area capacity to suit the current usage scenario dynamically. The experimental results show that both read and write performance of caching strategy based on SSD have improved a lot, especially for read performance. Compared with traditional caching strategy, the technique can be used in engineering to reduce write times to SSD and prolong its service life without lowering read-write performance.

1. Introduction

In recent years, CPU has been faster and faster, but storage system is always the performance's bottleneck of computer system. Meanwhile, in a computer system, I/O is still the biggest bottleneck. For some applications which have quite high requirements on response time, simply to improve CPU performance cannot improve the overall performance of a system; hence, it is necessary to improve the I/O performance and reduce the speed difference between storage system and CPU. However, caching technology is good enough to make up the gap between the performance of storage system and CPU. Through the analysis of locality of reference and accessing frequency, part of data will be stored in cache, greatly reducing the response time and improving the overall performance of system.

In order to solve the problem that the accessing speed of storage system cannot keep up with the processing speed of CPU, we brought in caching technology, whose mechanism is putting those frequently accessed data in fast accessing device, speeding up the access and reducing the latency time [1]. In the caching technology research, hot spot is still the policy to balance recency and frequency, while detection-based is the direction of the efforts, which combines those technologies in other fields of computer with cache replacement technology. Most of the current caching algorithms take into account locality of reference and accessing frequency; yet, it is still a big problem about how to combine the two factors to adapt to the current access features. Furthermore, each caching algorithm has its own emphasis and can only be applied to some sort of access pattern or service environment in using. But how to design a more versatile caching strategy

which can make dynamical adjustment according to different access patterns is an exciting research.

All traditional caching strategies take improving cache hit ratio as their main goal [2–4], an accurate method of evaluating for disks which have the same read-write cost. However, for the current SSD mainly based on flash media, read-write performance is greatly different, and, moreover, service time of SSD should be taken into account for the magnifying of write.

Thus, we should evaluate caching strategies for SSD by access cost [5, 6]. CFLRU is the first cache replacement algorithm, which applies LRU into flash medium storage and takes features of flash medium into consideration, prolonging write of dirty pages as far as possible and reducing write to flash medium. But there are some limitations because this strategy cannot make dynamical adjustment based on diverse access patterns and is greatly dependent on boundary definition and window size of working area and erasing area. So a new strategy is needed to include all the above [7–9].

With the emergence of SSD, it brings about traditional caching strategy with many new ideas because of its superior read-write performance. Nevertheless, SSD cannot spread universally for its fatal flaws: the high price and limit of erase-write times. Unfortunately, if we apply it to cache technologies in the future, these problems must be taken into consideration, and then SSD can play its best role [10–12]. Thus, there are still a lot of needs for research and exploration, such as how to apply SSD to cache technologies and how to combine SSD with features of other storage devices such as memory and disks. Different from disk-oriented caching strategy, caching strategy based on SSD does not take simply improving cache hit rate as its main target but needs to evaluate accessing overhead of system, reducing read-write response time as far as possible. In view of read-write characteristics of flash media and service life of SSD, we should reduce write times to SSD as much as possible without affecting the system performance. Besides, we need to design caching strategy that is suitable for different usage scenarios, and it can make dynamical adjustment according to different usage scenarios and read-write ratio, dropping accessing overhead to the minimum in the range that the system allows.

Flexible and adaptive cache strategy based on SSD-FAC strategy designed in this paper is a kind of caching algorithm based on SSD, the ultimate goal of which is to reduce response time of the system and the accessing overhead as far as possible and extend the service life of SSD when considering the limit of SSD erase-write times and without affecting the performance.

At the same time, it can make dynamical adjustment according to the current program access pattern and system response to adapt to the current usage scenario and achieve optimal performance.

The contributions of this paper are described as follows.

(i) In view of the analysis of some existing caching strategies, we abandon the design schema taking read-write speed as ultimate goal and propose design thought whose ultimate goal is to reduce accessing overhead, offering the basis for the design of FAC strategy, which can have ability of access patterns to make dynamical adjustment to adapt to the different scenes.

(ii) Research and design of the selection algorithm of hot data—DASH, which selects comparable amount of hot data each time according to the capacity of SSD. Because of the impact on system services caused by data migration, data migration does not process separately but processes with requests processing.

(iii) The result shows that the strategy can make dynamic adjustment according to the current access pattern and greatly reduce the write times to SSD and extend the service life of SSD while increasing the system performance of read and write. FAC cache is 13.5% higher than flash cache in sequential read request and 18.7% in sequential write, while it is 9.2% and 30.1% higher than flash cache in random read request and random write request averagely.

The rest of this paper is organized as follows. Section 2 shows related works. The design of FAC is introduced in Section 3. Section 4 gives the experimental evaluation and results. We make conclusions in Section 5.

2. Related Work

At present, many researchers have proposed flash-based buffer management algorithm for SSD usage scenarios, such as FAB [13], CLC [14], BPLRU [15], BPAC [16], BPCLC [17], CFLRU [18], and improved CRLRU [19]. Taking advantage of flash read-write performance asymmetry, CFLRU is a kind of buffer replacement strategy which first replaces read-only pages and assumes that write cost of flash is far greater than read cost. Its core idea is dividing LRU linked list into two parts: working area and replacement area. Once cache is full and some data needs to be replaced to outside, supposing there exists read-only data in replacement area, CFLRU will choose read-only pages for replacement according to LRU. When there are only dirty pages in replacement area, dirty pages at the tail of the linked list will be replaced. Some other researchers improved traditional LRU and LFU to accommodate diverse application requirements.

SieveStore [20] designed by Pritchett and Thottethodi and some others put forward two filter storage solutions based on high traffic, using a small SSD which can be shared to store hot data. In order to lower SSD write allocation selective buffer allocation strategy is adopted to make full use of cache. We can use two practical filtering methods to accurately find accessed frequently blocks. The first approach is an offline, discrete-allocation variant-SieveStore-D, which enforces selectivity using an access-count-based discrete batch-allocation (ADBA) mechanism. SieveStore-D maintains precise state of block, by logging each access, and periodically combines the states of block in an offline pass. Blocks are allocated into SSD only if their access count in an epoch exceeds a threshold value. The second approach is an online, continuous-allocation variant-SieveStore-C, which

sieves accesses using a hysteresis-based, lazy cache allocation mechanism wherein disk blocks are allocated into SSD on the n times (for some threshold n) miss over a recent time window. To maintain the state of blocks that are not cached, SieveStore-C uses a two-tier structure, a preliminary tier that sieves with imprecise (potentially aliased) state of block, followed by an accurate tier to maintain the quality of sieving.

Griffin [21] put forward by Soundararaja and some others uses disk as write cache of SSD, which can reduce write times to SSD in the case of high performance. Through tests, we found that Griffin can prolong the service life of SSD apparently and reduce the average I/O latency by 56%.

Within the industry, many companies put forward some solutions for SSD as cache and hierarchical storage medium.

SSD cache scheme of Huawei Symantec is called SmartCache. The scheme uses one or more SSDs as cache array, called SmartCache buffer pool. Acquire hot data statistics of other mechanical disks in the array and then every once in a while update to buffer pool; thus, when hot data is accessed, it can be directly read in SmartCache, which just takes enough advantage of high read performance of SSD, improving the read performance of the overall system. The scheme is mainly suitable for the case when there exist more read and less write hot data and mainly random small requests.

LSI launched LSI Mega RAID Cache Cade Pro2.0 read/write cache software used for some LSI Mega RAID 6 Gb/s SATA+SAS control cards, which uses SSD as a fast buffer to store hot data, improving read-write performance effectively, reducing I/O time delay, speeding up application response, and shortening RAID accessing and rebuilding time. This software can import hot data into SSD cache dynamically, with no need of artificially manual configuration.

Since 2009, EMC has successively launched two generations of FAST. The first generation of FAST is applicable to some storage series products, such as CLAR ii ON and Symmetrix. Function of FAST in different product lines is slightly different in details of setting and executing, however, the same on the basic modes of operation. In view of that, automatic migration of FAST is the whole LUN, and the granularity is not elaborate enough; thus, the layered effect is not optimal, and requirement of the source migration needs is too high. So far, FAST has been upgraded to the second generation, which is called FAST VP (short for virtual pool). As EMC announced, FAST VP adopts Sub-LUN level automatic migration technology, which means that, because of LUN, fidelity of data can go with Symmetrix VMAX and newly launched VNX to integrate storage products. In this paper, we have used the same idea to select the hot data and cold data.

3. FAC

3.1. The Architecture of FAC. During designing FAC caching algorithm, considering three relations among memory cache, SSD and disk, and their respective access features, FAC caching strategy can be divided into two modules: memory cache module and SSD cache module. The memory cache module is mainly applied to cache the hottest data, and according to the analysis of some accessing traces, we know that, in particular access pattern, only 4% data in average will be accessed in a large number, whose traffic takes over more than 90% of the total traffic, and these data will be stored in memory cache and attained directly when application needs. Besides, 3%–7% data belongs to hotter data and will be stored in SSD according to filtering mechanism of memory cache. The rest of the data will be directly written to the underlying disk due to its extremely low access frequency, avoiding the pollution to SSD caused by cold data. The architecture of FAC caching strategy is shown in Figure 1.

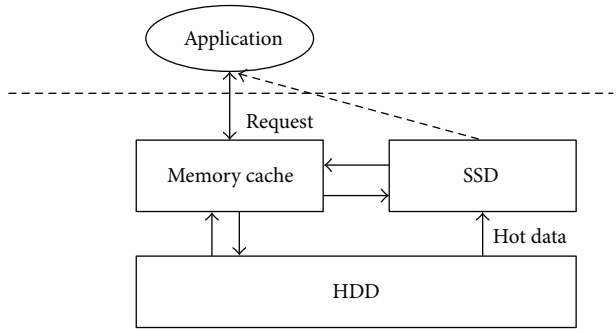

Figure 1: The architecture of FAC.

3.2. Design of Memory Cache Module

3.2.1. Data Organization. Based on read-write performance difference of SSD, data in memory cache can be divided into read area and write area. Data application read request is stored in read area and data write request is stored in write area. Hot data that has been recently accessed is stored both in read and write areas. If one of them is full, according to hot data filtering mechanism of memory cache, filtered hot data will be imported into SSD and cold data will be directly written into disk. As shown in Figure 2, for those data blocks obsoleted because of using up of free blocks in read area or write area, their Meta data will be preserved in respective memory block, convenient to subsequent analysis of access pattern and capacity adjustment of read and write area. Data blocks obsoleted in read and write area will immediately be written into disk. When data is confirmed to be hot data, it can be written into SSD. Once hot data in disk is imported into SSD, Meta data zone which is mainly used to record Meta data of SSD needs to be updated, including the mapping relationship between data in SSD and data in disk. In addition, data consistency of disk and SSD needs to be recorded by Meta data zone.

In view of write speed of SSD and depletion to SSD caused by frequent writing, adopting hot data filtering mechanism can as far as possible avoid the case that cold data that has been stored in SSD directly is obsoleted into disk because of none access, causing pollution to SSD. Therefore, data that has been confirmed to be hot data through hot data filtering mechanism can be written into SSD.

3.2.2. Elimination Strategy. Through review of some currently mainstream cache replacement algorithm, class LRU

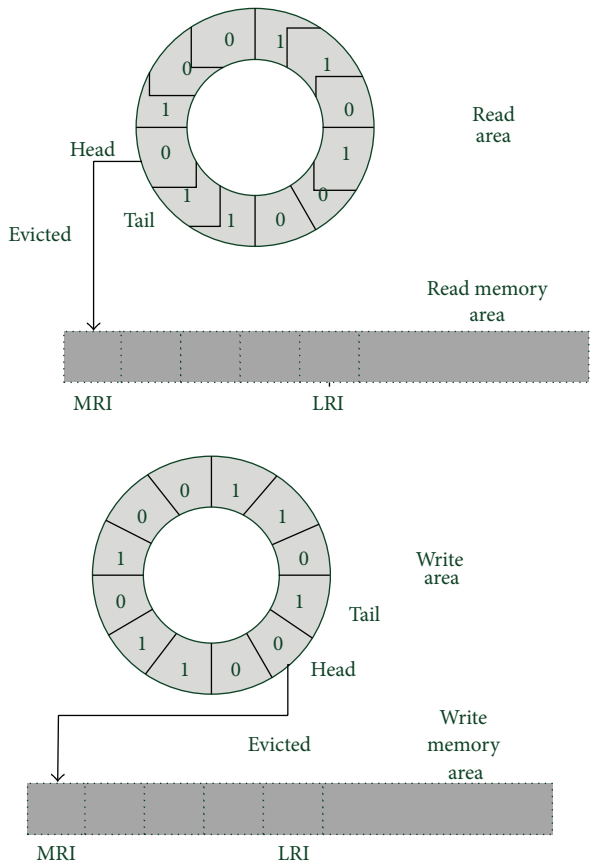

Figure 2: Data elimination in read-write area.

algorithm is suitable for most usage scenarios. However, LRU needs to record related access record terms, and each time these record terms are accessed, they need to be updated and large amount of extra space is also needed, and the cost of operation is great. But, by attentive analysis, according to the features of locality of reference, there is no need to record accurately the access situation of each data block but only to judge whether the data block in the window at the present has been accessed. In this way, cost of space and time can be significantly reduced. Therefore, we adopt CLOCK replacement algorithm.

The main idea of the algorithm is that an access bit is set in every page, and all pages in memory are linked into a circular queue through a link pointer. When one page is being accessed, its access bit will be set to 1.

When the replacement algorithm chooses one page to eliminate, it only needs to check the access bit of the page. If the bit is zero, then the page will be replaced; if the bit is one, then the bit will be set to zero again, and the page will not be replaced for the time being and will have the second chance to stay in memory. Then, the next page will be checked according to FIFO algorithm. Once it comes to the last page of the queue, if the access bit is still one, then it goes back to the head of queue to check the first page, as shown in Figure 2.

On the basis of hot and cold data selection algorithm, the cold data which were eliminated from read-write memory area would be directly written to disk instead of entering SSD. Meta data of those are kept in cold data memory area which shares the feature with read-write memory area.

3.2.3. Dynamic Adjustment Mechanism of Read-Write Area. When an application write request is not required hitting from disk or SSD in the first reading is needed; there may be a large part because of the need to phase out dirty blocks to operate the disk writing, so that the missing of writing causes much more space. When reading misses, if the selection algorithm of thermal SSD data can select a large part of hot spot data, you can make most of the read requests completed in the SSD, apparently costing very little space. According to the current needs of read-write access mode and the characteristics of the system, the read-write in the memory buffer zone is dynamically adjusted to minimize the cost of accessing to the system, which makes the system applicable in the current scenario optimal.

Because large performance difference of flash media always causes cost difference of access, traditional method that evaluates replacement algorithm based on hit rate is not applicable anymore; instead, cost of access needs to be taken as the ultimate goal. FAC strategy divides the memory cache into read area and write area, and capacity of write area is 8 times larger than that of read area.

Shown as follows, the adjustment mechanism of read-write area based on access pattern is that, assuming that the total capacity of memory buffer is S, capacity of read area is $1/9 \times S$ and that of write area is $8/9 \times S$.

Considering that Meta data of blocks that have recently been eliminated is preserved in read-write memory area, when read request does not hits in read area but hits in read memory area, it shows that the data has been accessed not long ago, but the block was replaced by updated block because of the capacity of read area. Therefore, we take into account the increasing capacity of read area. Every time one unit length is added, capacity of write area will decrease by one unit length at the same time, keeping the total length invariant. It is in the same way for write request of application. Because capacity of write area is 8 times larger, there are two strategies for increasing write area. One is gradually changing, which means a minor increase every time. For example, every time two units length is added and the read area decreases the same length meanwhile. If the effect is not so obvious, increasing length will be 4 units length, 8 units length, and so forth. Different from gradual changing, leaping means that increasing length will be on the basis of the capacity rate between write area and read area, which is to say 8 units length will be added to write area each time. Leaping strategy can help achieve ideal effect rapidly; however, it also is easy to cause sharp decreasing of read area capacity and increasing of hit rate of read memory area, and the case cannot be improved immediately because of the restriction of expansion unit length of read area. We give some definitions in Table 1.

Expansion of read area:

$$\begin{aligned} \mathrm{RAC\_}(t\_i) &= \mathrm{RAC\_}(t\_(i-1)) + P, \\ \mathrm{WAC\_}(t\_i) &= \mathrm{WAC\_}(t\_(i-1)) - P. \end{aligned} \tag{1}$$

Table 1: Symbols and meaning.

Symbol	Meaning
P	Unit length
RAC	Read area capacity
WAC	Write area capacity
i_m	The minimum hot spot rank
N_b	The total access frequency
HR(t)	Hot spot ratio
HR_m	The minimum hot spot ratio
K_i	Time window
H_B	Heat counting
δ_i	Weight of K_i window
K	Access times of the current time window

Expansion of write area:

$$\begin{aligned} \mathrm{WAC}\_(t\_i) &= \mathrm{WAC}\_(t\_(i-1)) + P', \\ \mathrm{RAC}\_(t\_i) &= \mathrm{RAC}\_(t\_(i-1)) - P'. \end{aligned} \tag{2}$$

P' increases progressively from P to $2P$, $4P$, and $8P$:

$$\begin{aligned} \mathrm{WAC}\_(t\_i) &= \mathrm{WAC}\_(t\_(i-1)) + 8P, \\ \mathrm{RAC}\_(t\_i) &= \mathrm{RAC}\_(t\_(i-1)) - 8P. \end{aligned} \tag{3}$$

3.3. Design of SSD Cache Module

3.3.1. Selection Algorithm of Hot Data. First, let us see some existing research in this problem. Some researchers have put forward OHPM (short for object hot spot prediction model) to serve for hot spot data selection. In a storage system, N objects form a set S, in which objects are in order, and o_i means object ranking the ith place. The total access frequency of all objects is M and hot spot queue length is N_b. At a time T, assuming an object o_i in S is in hot spot queue, at t which equals k times of T, if o_i ranks the i'th place, then what is the value of i'?

After some researches, the minimum hot spot rank can be counted by the following:

$$i_m = N_b k^{-1/\alpha}. \tag{4}$$

And hot spot ratio is confirmed by the following formula:

$$\mathrm{HR}(t) = \frac{H(t)}{N_b}. \tag{5}$$

Further, the minimum hot spot ratio can be conducted by the previous formula and some other tests. Consider

$$\mathrm{HR}_m = k^{-1/\alpha}. \tag{6}$$

Selecting cold and hot data is mainly judged by its access times; when it exceeds a specified threshold, the data will be judged as hot data, and the threshold can make dynamical adjustment. It is unnecessary to set a counter for every data block because of the huge cost. Through the observation on the access trace of a large number of applications, only about 2% data can be judged as hot spot data in the absolute sense that will be accessed by a large number of applications. 4%–7% data is counted as hotter data because of relatively high access times. The rest of data has quite a little access time. On account of this point, we proposed selection algorithm of hot and cold data, Dynamical Adjust Section Hot data algorithm, called DASH, which is designed in two-layer structure. The first layer is filter layer, filtering blocks that do not achieve the threshold. Before filtering, multiple blocks are mapped into a large virtual data block, set with a counter as shown in Figure 3. For instance, 100 blocks are mapped into virtual data block, and those data blocks that have exceeded the filter value are qualified to enter the second layer of accurate statistics. For most filtered virtual data blocks, it is dispensable to count access times of each request page because the traffic of every virtual data block is less than the threshold. The second layer is statistics layer, counting accurately the access times of each small data block or request page and further judging whether it is hot data or not. In the procedure of implementation, the number of data blocks to make up a virtual data block depends on the characteristics of data stored in the current system. The less the amount of hot data is, the larger the request page selected should be; then, the space for counting hot data will be smaller. And many misjudgments caused by small request page in large data block, which will lead to entering the second layer of large amount of cold data, can be avoidable because of less hot data amount. On the contrary, in the case of plenty of hot data, selecting a hand of request pages to form large data block can avoid entering the second layer of large amount of cold data because of too many request pages in large data block and the counting cost of time.

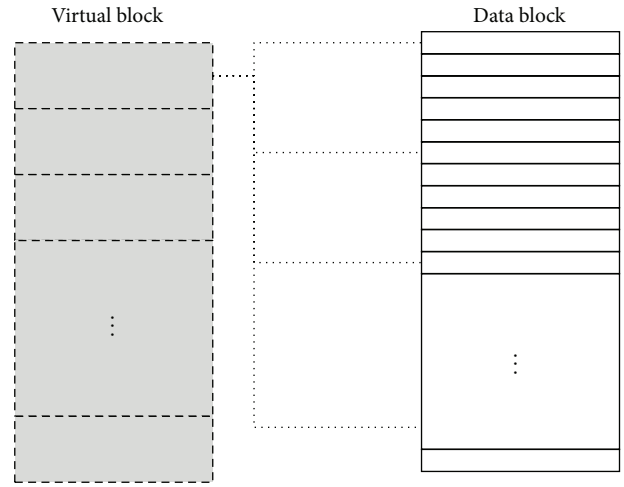

Figure 3: Multiple-one mapping.

The virtual blocks whose traffic exceeds filter value will generate an accurate statistical table, called Hash Table, which is used to record later access times of each request page in the virtual block as shown in Figure 4. The virtual block which has generated an accurate statistical table will enter the second layer to conduct accurate statistics, mainly, selecting a time window first and then counting the access times of

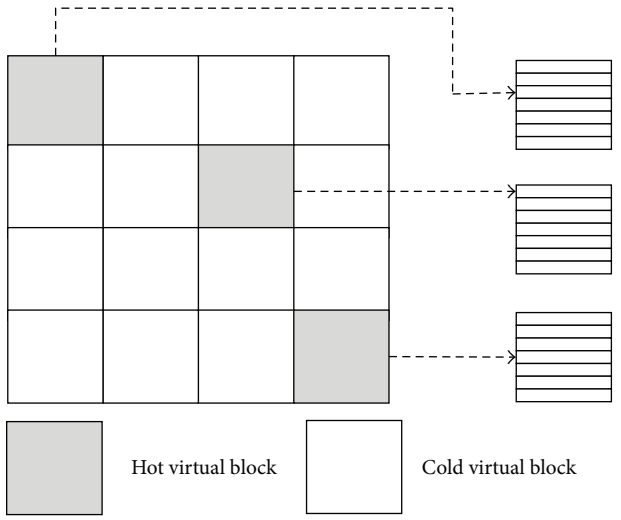

Figure 4: Multiple-one mapping.

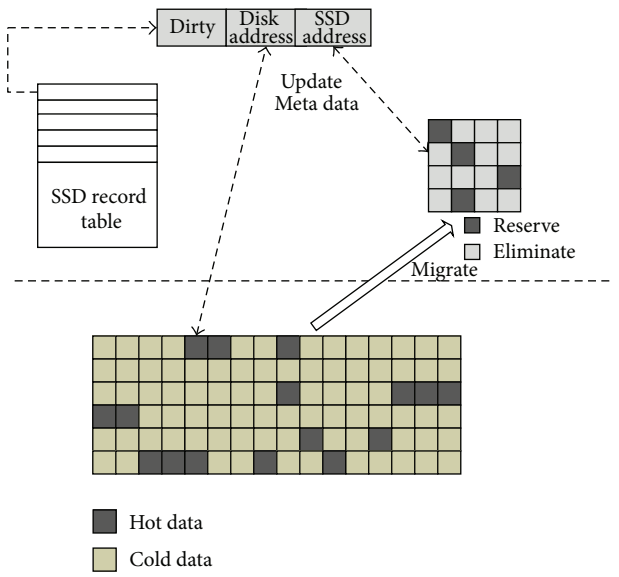

Figure 5: Hot data migration.

each page in the time window. The time window will be divided into several smaller time windows such as K, for the characteristics of locality of reference. Access times weight of each time window is different; the closer to present time the time window is, the larger the weight is.

The heat of each request block in the whole time window can be counted according to the access times of request block and weight of each time window. Assuming that the weight of K_i window is δ_i, then heat counting formula of the current request page is

$$H_B = \sum_{i=1}^{N} K_i \delta_i. \tag{7}$$

For one data block, counting formula of its access times is

$$K_{T_i} = K + \delta K_{T_{i-1}}. \tag{8}$$

K refers to the access times of the current time window.

3.3.2. Hot Data Migration. In view of hot data updating with every data access, only when the data block accessed in disk is confirmed to be hot data before the next access and is marked as "ought to be in SSD," it can be written into SSD. Thus, it is necessary to set a record table to record the mapping relation between data in SSD and that in disk and update the record table at the same time of updating data in SSD. Besides, because of write request of application not passing SSD and all data update completed in disk, which might lead to inconformity of data in disk and SSD, it is needful to set a flag bit to record whether the data in disk and SSD is consistent or not.

Before writing new hot data into SSD, we need to select one data block in SSD to eliminate. Considering that flag bit of data in SSD has been modified after hot data filtering, for those data blocks without being marked with "reserving," they can be replaced. After modifying flag bit, the address of these data blocks can be recycled, similar to memory management mechanism. Forming a linked list with data blocks waiting to be eliminated, when elimination is needed, one data block in the list will be directly eliminated and SSD record table will be updated, as shown in Figure 5.

3.3.3. Dirty Data Migration. When it comes to the write request of application, those data not hit in memory cache will be directly read into memory from disk or SSD and then be modified. Before data page is read into memory, it will involve elimination in the case of that memory is full, and dirty block which is eliminated will be written into disk; thus, when writing data block including SSD copy, it leads to data inconformity of disk and SSD. So, dirty data migration mechanism is needed to ensure the consistency.

One policy is that, when modifying data including SSD copy, a flag bit is set to mark the data block, which can be only read from disk if it is dirty. That is to say, for those hot data that have been modified, SSD does not work and data in SSD has been invalid. Data marked as dirty block can be imported into SSD according to a particular policy, mainly guiding disk by time and according to the threshold. In the process of guiding disk by time, start guiding disk generator at set intervals, and update dirty data blocks in disk into SSD; meanwhile, update Meta data flag bit in SSD. In practical applications, frequency of guiding disk depends on the access features of the current system. If guiding disk is too frequent, it is likely to cause repetitive writing of SSD, reducing the service life of SSD. However, once the interval of guiding disk is too long, then maybe large amount of hot data needs to be accessed in disk, affecting the performance of the system. The other policy is guiding disk by threshold, meaning that the operation of guiding disk will be launched when dirty blocks in disk reach a certain quantity and write dirty data into disk without affecting disk. Magnitude of threshold is decided according to both the current access features and condition of loading.

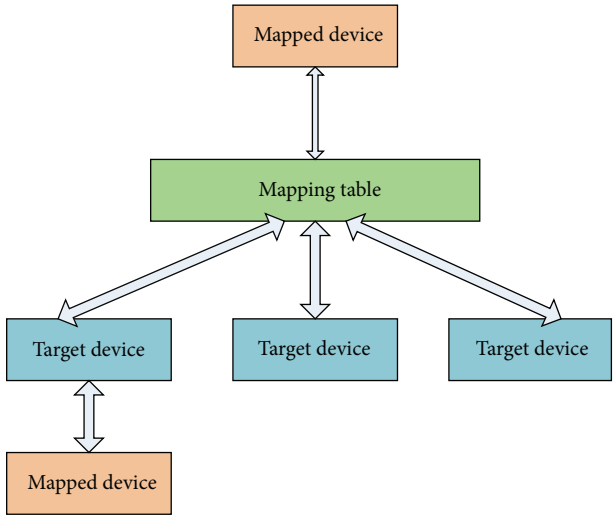

Figure 6: Device mapper mechanism.

Another strategy is that, if the coming write request hits in SSD, modify data in SSD directly. Compared with the previous strategy, the benefit of this strategy is to reduce system overhead and lower response time due to the faster write of SSD. Defects come with meeting frequent write, leading to multiple repetitive write and affecting the service life of SSD. Data modified in SSD, needs to be marked as "dirty" and guided into disk at regular time, enduring the consistency of data in disk and SSD.

4. Implement and Evaluation Methodology

4.1. Implements. FAC strategy designed in this paper based on SSD is implemented on the basis of flash cache. Flash cache is a new open source project of Facebook technical team, the fundamental purpose of which is using SSD to cache data to accelerate a kernel module of MySQL. It was originally used to accelerate the database and meanwhile was designed as general caching strategy, applicable to any applications built up on block device.

Based on Device Mapper mechanism, SSD and ordinary disk are mapped into a logical block device with cache, to be the interface of user action. Users perform read and write operations directly not on underlying SSD or ordinary disk but on this logical device. Which operating on these underlying block devices, caching function offered as an entirety will be lost.

Mapped device builds up mapping relations through mapping table and three target devices, and target device could be a single storage device or be evolved through mapped device and build up mapping relation through mapping table and the next layer target device as shown in Figure 6.

4.2. Experimental Setup. Cache system based on SSD works mainly on the existing storage server in laboratory, which is linked to server through client and tested with its read and write performance in client-side. The environmental hardware configuration of server and client is provided in Tables 2 and 3.

Table 2: Server hardware environment.

CPU	Intel Xeon CPU E5606 @ 2.13 GHz 8 cores
RAM	DDR2 16 G
DISK	Seagate 7200 1 T
SSD	Intel SSD 320 Serials 120 G
OS	RHEL5.4 x86_64
KERNEL	Linux2.6.18-164

Table 3: Client machine environment.

CPU	Intel Pentium Dual-Core CPU E5300 @ 2.60 Hz
RAM	DDR2 2 G
DISK	Seagate 7200 500 G
OS	Windows 7 Ultimate 32 bit

The test mainly adopts Iometer to test the read-write performance of the system. Iometer is an I/O subsystem measurement and a characterization tool for single and clustered systems created by Intel Corporation, which can test performance of disk or network controller, capability of bandwidth transmission and response, network throughput of devices, and performance of hardware and network. Iometer primarily consists of Iometer and dynamo. Launch dynamo on the storage server first, link it to client machine, and then turn on Iometer application on client, and so we can see server-side host and disk to test in the case of different access patterns.

4.3. Performance Test Results

4.3.1. Read and Write Speed Comparison in Sequence. Performance test for caching strategy based on SSD on hybrid storage system is mainly to test read-write performance of system, capability of making dynamical adjustment based on access pattern, and depletion to SSD. We test depletion to SSD mainly by counting write times to SSD and judging and then comparing it with caching strategy of flash cache itself.

It can be seen from Figure 7, compared with included LRU caching algorithm of flash cache, in sequence, that FAC does not have performance advantage at the beginning, but its advantage of read-write performance has surfaced for making dynamical adjustment with the handing out of read-write requests. Furthermore, owing to SSD being mainly used as read cache in FAC caching strategy, it can be told from the figure that increasing read speed is larger and write speed do not decrease but suffer a certain growth.

While focusing on sequential read request, when block requests 8 KB, read speed of FAC has been obviously superior to LRU caching algorithm of flash cache, and with the processing of request handing, the advantage becomes more and more visible. When block requests 128 KB in sequential write request, read speed of FAC caching strategy starts to

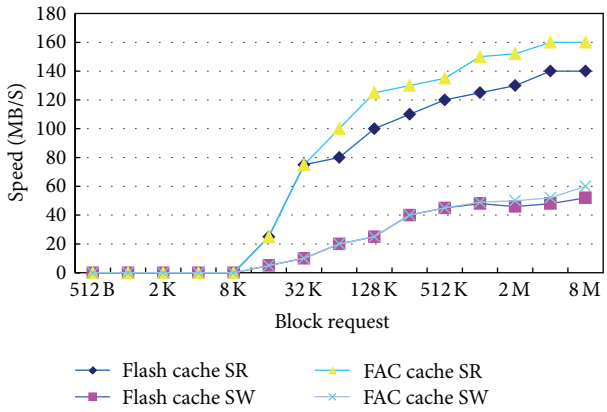

Figure 7: Read-write performance comparison between flash cache and FAC in sequence.

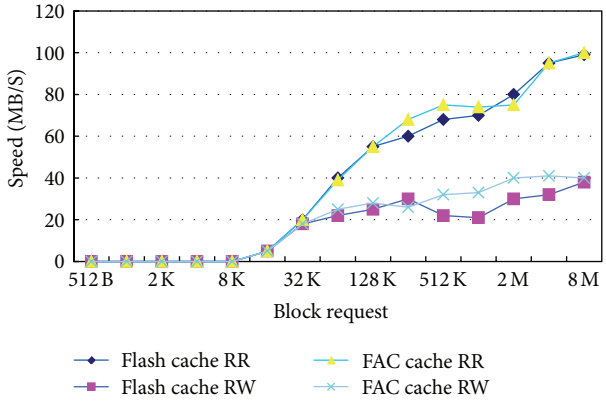

Figure 8: Read-write performance comparison between flash cache and FAC in random.

exceed that of flash cache and then superiority becomes larger and larger.

4.3.2. Read and Write Speed Comparison in Random. It can be seen from Figure 8, because of low cache hit rate in random, read-write performance of both flash cache LRU algorithm and FAC caching strategy is far lower than that in sequence. However, compared with flash cache LRU algorithm, FAC has a certain performance advantage in random even though the increasing is inferior to that in sequence.

It can be told from the figure that performance advantage of FAC caching strategy begins to appear from block size being 128 KB in case of random. Though its speed waves are slower than that of flash cache LRU algorithm when block size reaches 2 M, the overall trend of curve is still superior to flash cache LRU. In the case of random write, when block size is 32 KB, write speed of FAC caching strategy starts to exceed that of flash cache LRU algorithm and the overall trend is always superior to flash LRU algorithm.

The encoding efficiency with ICRS coding is 34.2% higher than using CRS and 56.5% more than using RS coding equally. The decoding rate by using ICRS is 18.1% higher than using CRS and 31.1% higher than using RS averagely.

5. Conclusions

FAC caching strategy based on SSD and cloud platform is designed in this paper. The strategy is combined with designing ideas of traditional caching strategy at the same time of giving full consideration to the characteristics of SSD itself, maximizing the role SSD plays. It can be seen from the test result that, benefit from the capability of making dynamical adjustment of FAC caching strategy and the advantage SSD has as read cache, the system can make dynamical adjustment according to the access pattern, greatly reduce write times to SSD, and prolong its service life without lowering read-write performance.

Caching strategy designing scheme designed in this paper is able to reduce the response time of system and the erase times to SSD. After that, considering that capability of making dynamical adjustment of memory cache would do little to make read-write of large disk qualitative change, it will be the next research target to combine memory cache and SSD better to improve accessing performance of disk.

So far, we have just implemented capability of making dynamical adjustment of memory cache module and hot data selection algorithm and data migration of SSD module, and the next target is to implement prefetching in system to improve performance.

References

[1] B. S. Gill and L. A. D. Bathen, "Optimal multistream sequential prefetching in a shared cache," *ACM Transactions on Storage*, vol. 3, no. 3, article 10, 2007.

[2] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proceedings of the USENIX Symposium on Internet Technology and Systems*, 1997.

[3] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[4] C. Dirik and B. Jacob, "The performance of pc solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*, pp. 279–289, June 2009.

[5] L. Youyou, S. Jiwu, and Z. Weimin, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)*, pp. 257–270, 2013.

[6] T. Kgil and T. Mudge, "FlashCache: a NAND flash memory file cache for low power web servers," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*, pp. 103–112, October 2006.

[7] G. Wu and X. He, "Reducing SSD read latency via NAND flash program and erase suspension," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, 2012.

[8] Y. Oh, E. Lee, D. Lee, J. Choi, and S. H. Noh, "Hybrid solid state drives for improved performance and enhanced lifetime," in *Proceedings of 29th IEEE Symposium on Massive Storage Systems*, 2013.

[9] S. Lee, K. Ha, K. Zhang, J. Kim, J. Kim, and F. S. Flex, "A flexible flash file system for MLC NAND flash memory," in *Proceedings of the USENIX Annual Technical Conference*, San Diego, Calif, USA, June 2009.

[10] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proceedings of the USENIX on Annual Technical Conference (ATC '08)*, 2008.

[11] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND flash based disk caches," in *Proceedings of the 35th International Symposium on Computer Architecture (ISCA '08)*, pp. 327–338, June 2008.

[12] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridStore: a cost-efficient, high-performance storage system combining SSDs and HDDs," in *Proceedings of the 19th Annual IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '11)*, pp. 227–236, July 2011.

[13] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, "FAB: flash-aware buffer management policy for portable media players," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 2, pp. 485–493, 2006.

[14] S. Kang, S. Park, H. Jung, H. Shim, and J. Cha, "Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices," *IEEE Transactions on Computers*, vol. 58, no. 6, pp. 744–758, 2009.

[15] H. Kim and S. Ahn, "BPLRU: a buffer management scheme for improving random writes in flash storage abstract," in *Proceedings of 6th USENIX Conference on File and Storage Technologies*, pp. 158–171, ACM Press, San Jose, Calif, USA, 2008.

[16] G. Wu, B. Eckart, and X. He, "BPAC: an adaptive write buffer management scheme for flash-based solid state drives," in *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST '10)*, IEEE Computer Society Press, Incline Villiage, Nevada, May 2010.

[17] H. Zhao, P. Jin, P. Yang, and L. Yue, "BPCLC: an efficient write buffer management scheme for flash-based solid state disks," *International Journal of Digital Content Technology and its Applications*, vol. 4, no. 6, pp. 123–133, 2010.

[18] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, "CFLRU: a replacement algorithm for flash memory," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*, pp. 234–241, ACM Press, October 2006.

[19] H. Shim, B.-K. Seo, J.-S. Kim, and S. Maeng, "An adaptive partitioning scheme for DRAM-based cache in solid state drives," in *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST '10)*, May 2010.

[20] T. Pritchett and M. Thottethodi, "SieveStore: a highly-selective, ensemble-level disk cache for cost-performance," in *Proceedings of the 37th International Symposium on Computer Architecture (ISCA '10)*, pp. 163–174, ACM Press, June 2010.

[21] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD lifetimes with disk-based write caches," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10)*, 2010.